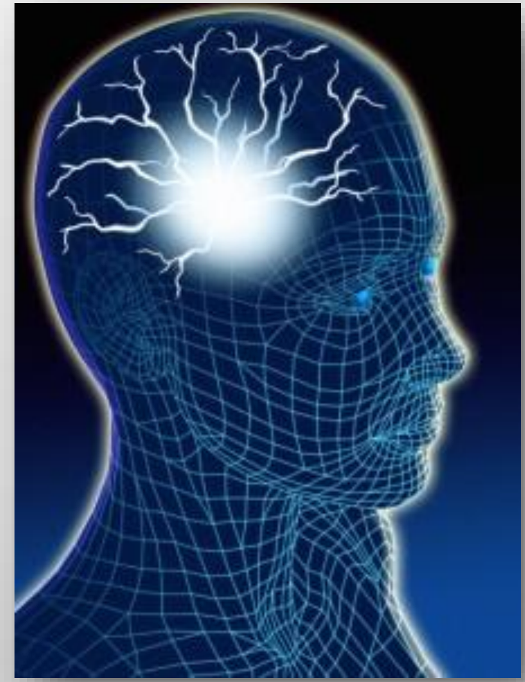
Solving problems by searching

Chapter 3



NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Announcements

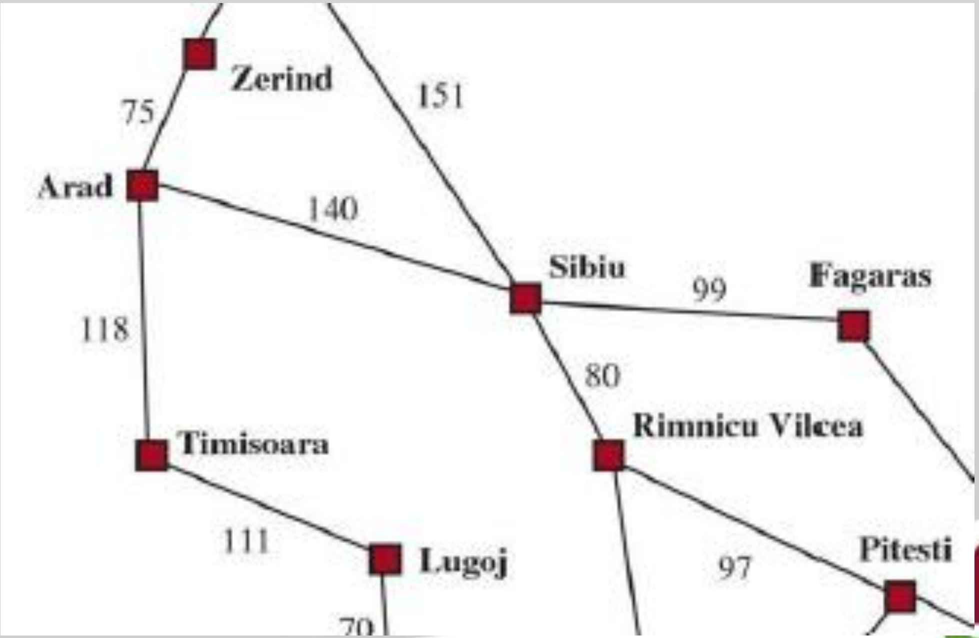
- Some people use both the third and fourth edition of the AIMA textbook
- Please consult only the fourth edition as you will be assessed on this edition

Lecture outline

- Uninformed search strategies
- Study unit 3 – Problem solving with the aid of search methods
- At the end of the lecture:
  - Understand uninformed search methods and then apply these methods to problems

Uninformed search strategies

- An uninformed search algorithm is given no clue about how close a state is to the goal(s)
- In contrast, an informed agent who knows the location of each city knows that Sibiu is much closer to Bucharest than Zerind and thus more likely to be on the shortest path



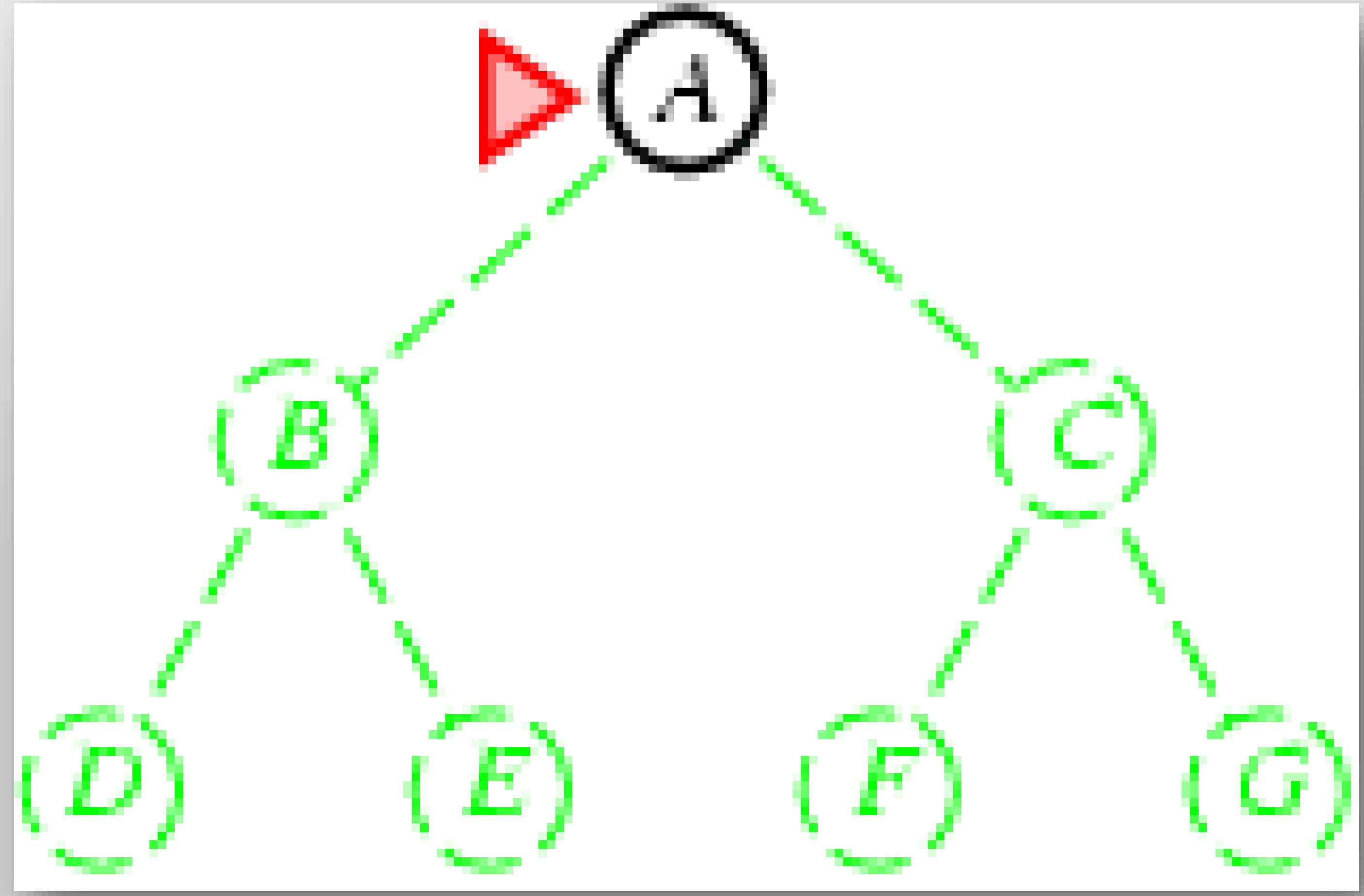
Breadth-first search

- When all actions have the same cost, an appropriate strategy is breadth-first search
- The root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on
- This is a systematic search strategy that is therefore complete even on infinite state spaces

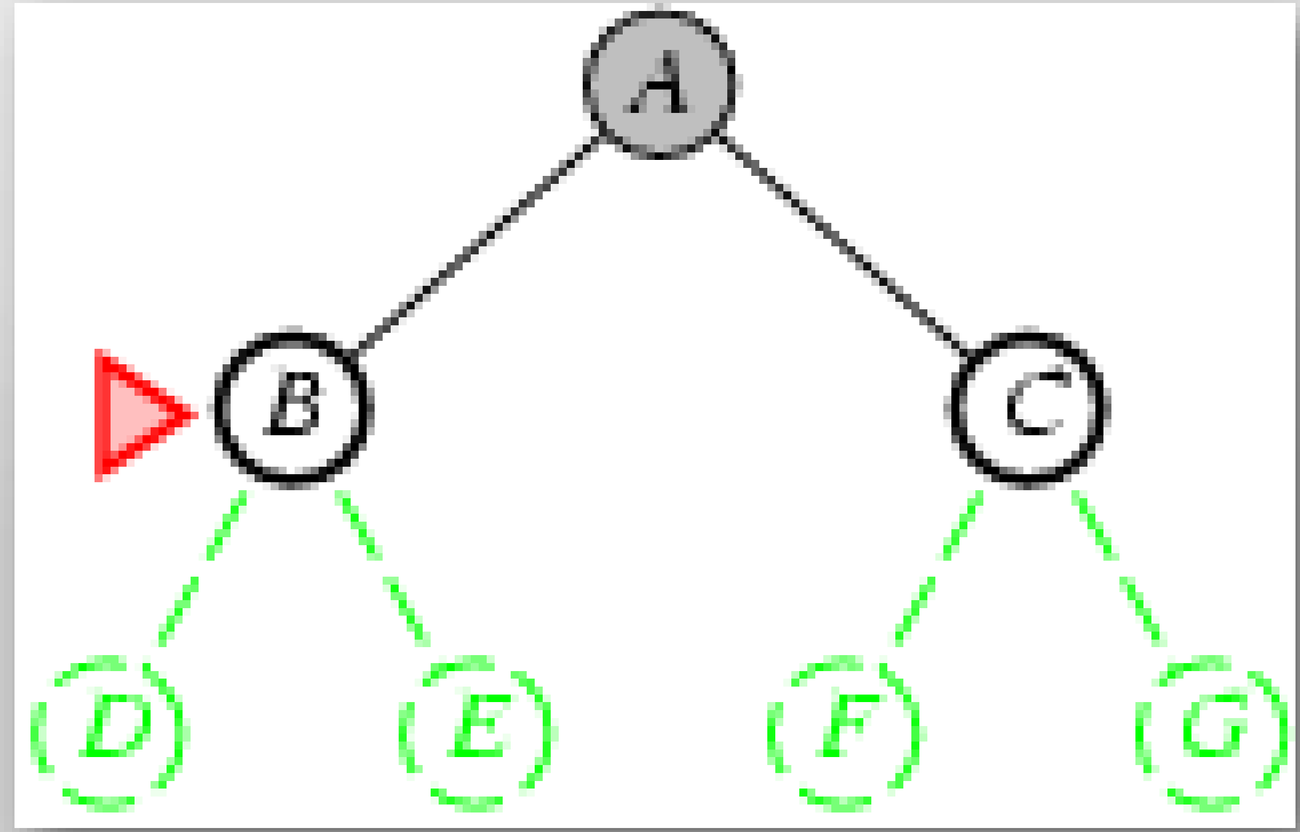
Breadth-first search

- Breadth-first search can be implemented where the evaluation function is the depth of the node — that is, the number of actions it takes to reach the node
- A couple of tricks will provide additional efficiency

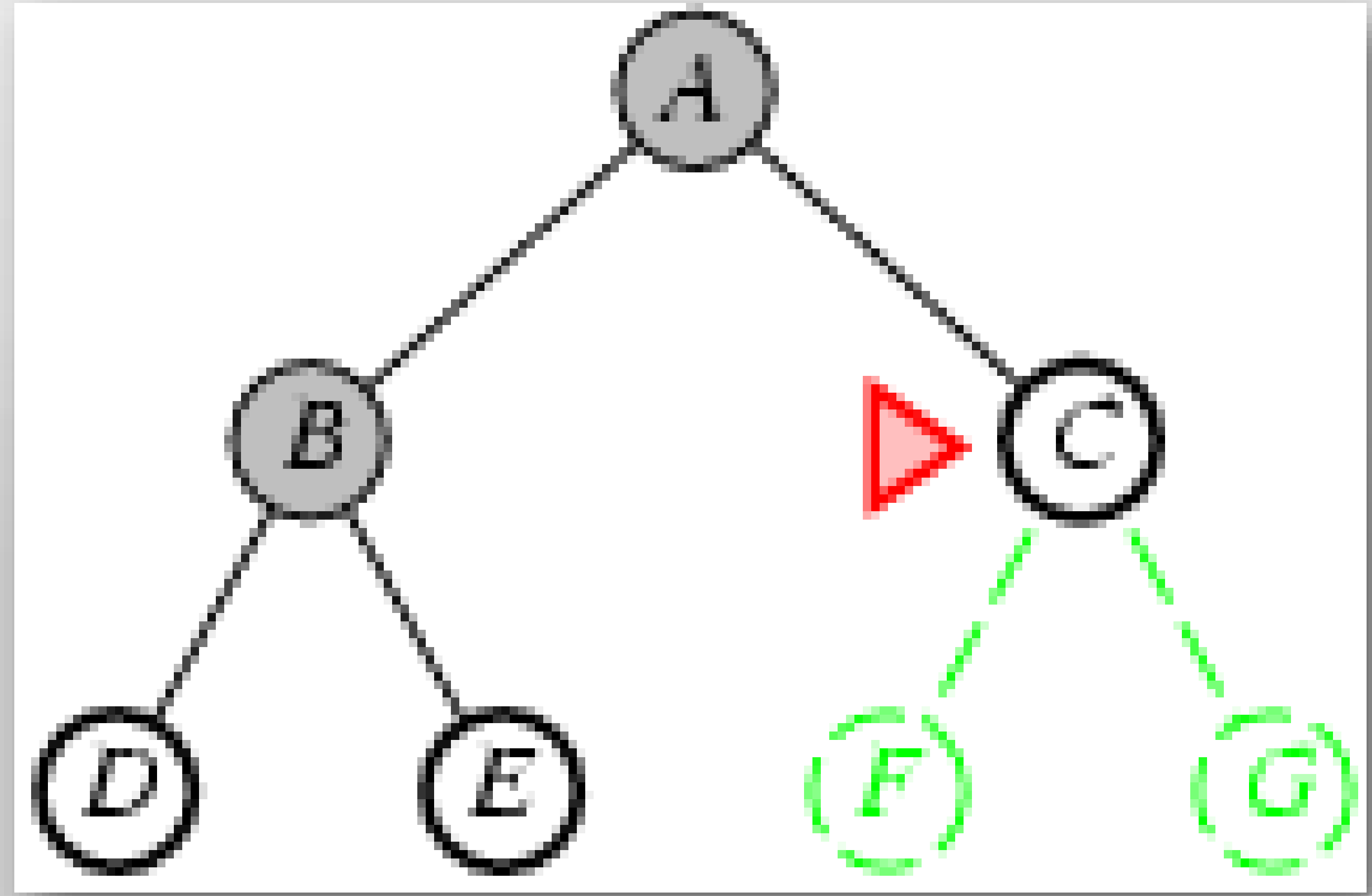
Breadth-first search



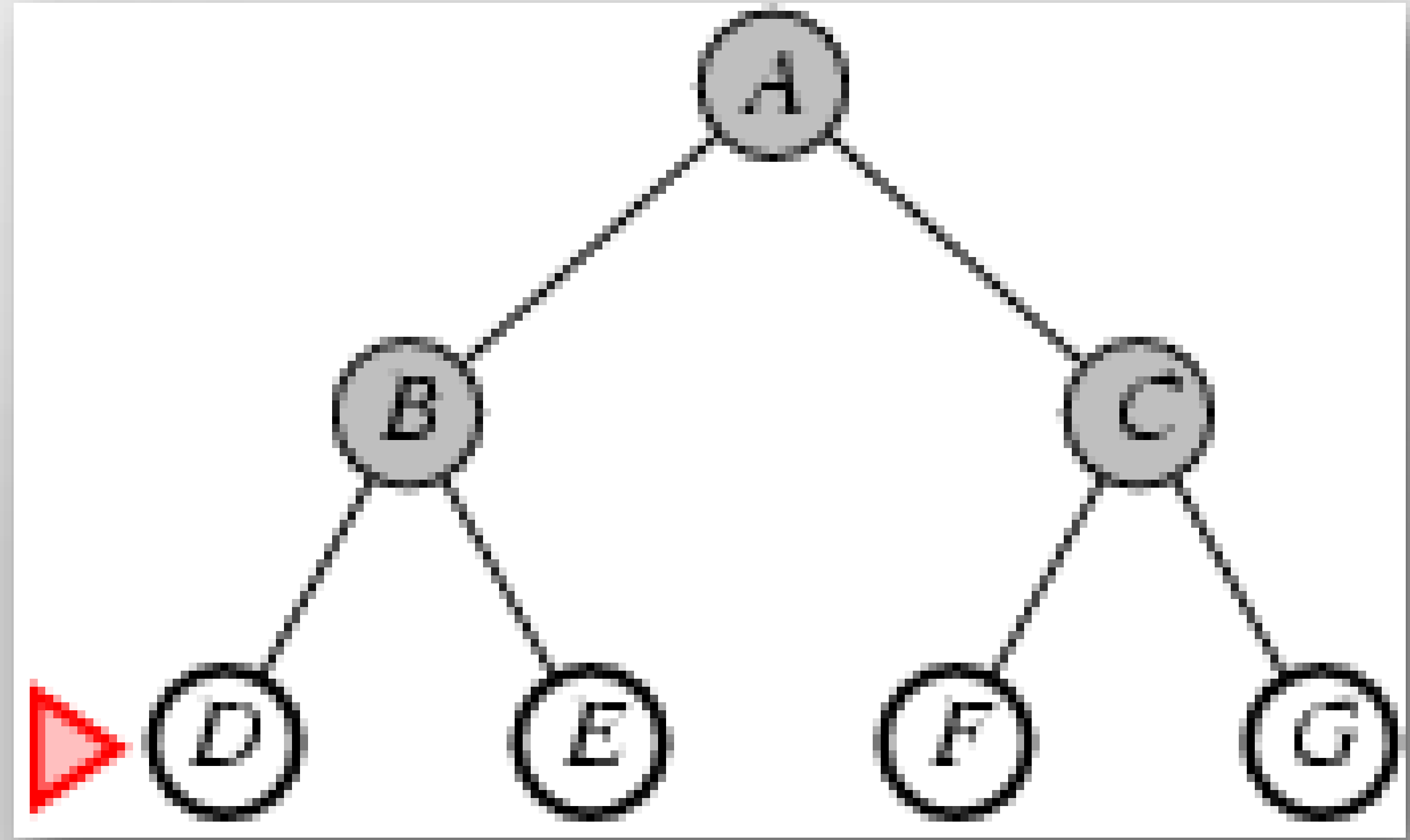
Breadth-first search



Breadth-first search



Breadth-first search



Breadth-first search

- Breadth-first search always finds a solution with a minimal number of actions, after nodes generated at depth *d* it has already generated all the nodes at depth *d – 1* so if one of them were a solution, it would have been found
- It is cost-optimal for problems where all actions have the same cost, but not for problems that don’t have that property
- It is complete in either case

Breadth-first search

- Breadth-first search always finds a solution with a minimal number of actions, after node … pth *d* it has already gene … s at depth *d* – *1* so if on … solution, it would have been found
- It is cost-optimal for problems where all actions have the same cost, but not for problems that don’t have that property
- It is complete in either case

Does it find a solution with the lowest path cost of all solutions?

Breadth-first search

- Breadth-first search always finds a solution with a minimal number of actions, after nodes generated at depth *d* it has already generated all the nodes at depth *d – 1* so if one of them were a solution, it would
- It is cost-optimal f all actions have the s for problem that don't have that property
- It is complete in either case

Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?

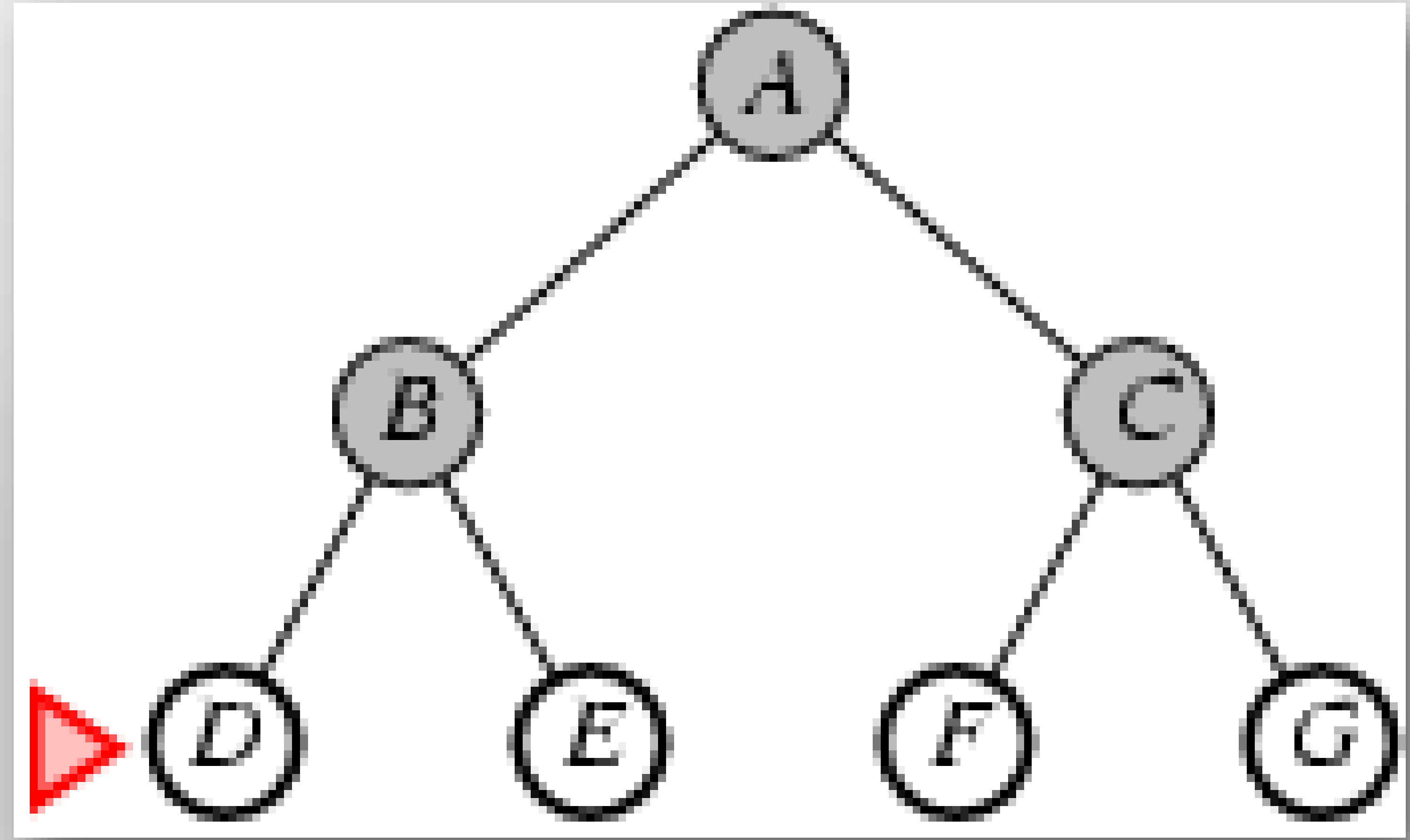
Breadth-first search

- Space and time complexity
  - In a uniform tree, each state has *b* successors
  - Root has 1 node, then *b* nodes, then b^2 nodes, then b^3 nodes, etc.
  - Suppose solution is on depth *d*
  - Total number of nodes generated is

$$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

  - Each node that is generated must be stored in the memory
  - Space complexity the same as time complexity

Breadth-first search



Breadth-first search

- As a typical real-world example, consider a problem with branching factor $b = 10$, processing speed 1 million nodes/second, and memory requirements of 1 Kbyte/node
- A search to depth $d = 10$ would take less than 3 hours, but would require 10 terabytes of memory
- Memory requirements a bigger problem as execution time
- Time requirement still a problem

Breadth-first search

- At depth $d = 14$, even with infinite memory, the search would take 3.5 years
- In general, exponential complexity search problems cannot be solved by uninformed search for any but the smallest instances

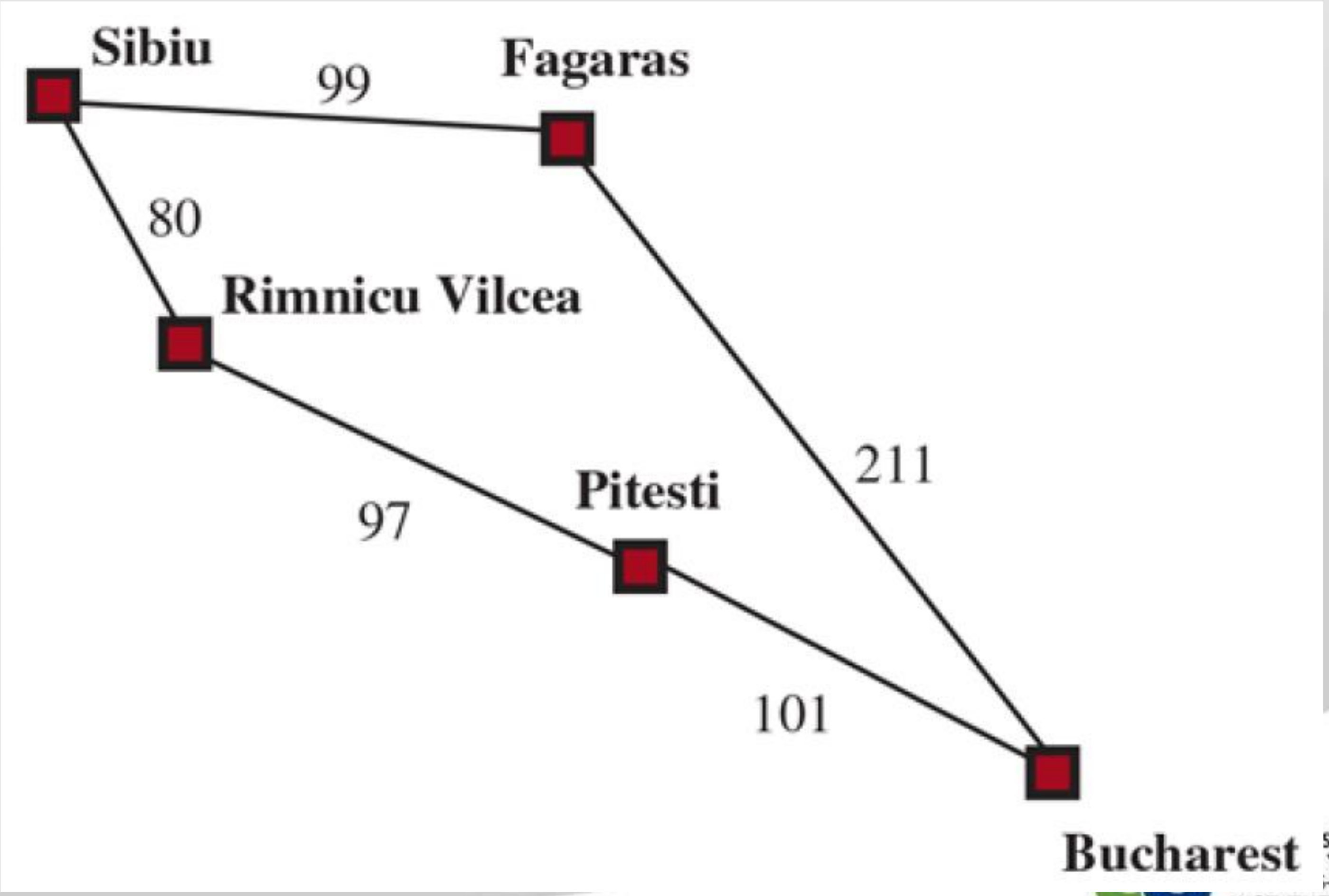
Dijkstra’s algorithm or uniform-cost search

- When actions have different costs, an obvious choice is to use best-first search where the evaluation function is the cost of the path from the root to the current node
- The idea is that while breadth-first search spreads out in waves of uniform depth—first depth 1, then depth 2, and so on — uniform-cost search spreads out in waves of uniform path-cost

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Dijkstra’s algorithm or uniform-cost search

- Consider this figure, where the problem is to get from Sibiu to Bucharest



Dijkstra’s algorithm or uniform-cost search

- The complexity of uniform-cost search is characterized in terms of C*, the cost of the optimal solution, and ε, a lower bound on the cost of each action, with $\varepsilon > 0$
- The algorithm’s worst-case time and space complexity is $O(b^{1+\lfloor C*/\varepsilon \rfloor})$ which can be much greater than b^d
- When all action costs are equal, $b^{1+\lfloor C*/\varepsilon \rfloor}$ is just b^{d+1} and uniform-cost search is similar to breadth-first search

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Dijkstra’s algorithm or uniform-cost search

- Uniform-cost search is complete and is cost-optimal, because the first solution it finds will have a cost that is at least as low as the cost of any other node in the frontier
- Uniform-cost search considers all paths systematically in order of increasing cost, never getting caught going down a single infinite path (assuming that all action costs are $> \varepsilon > 0$)

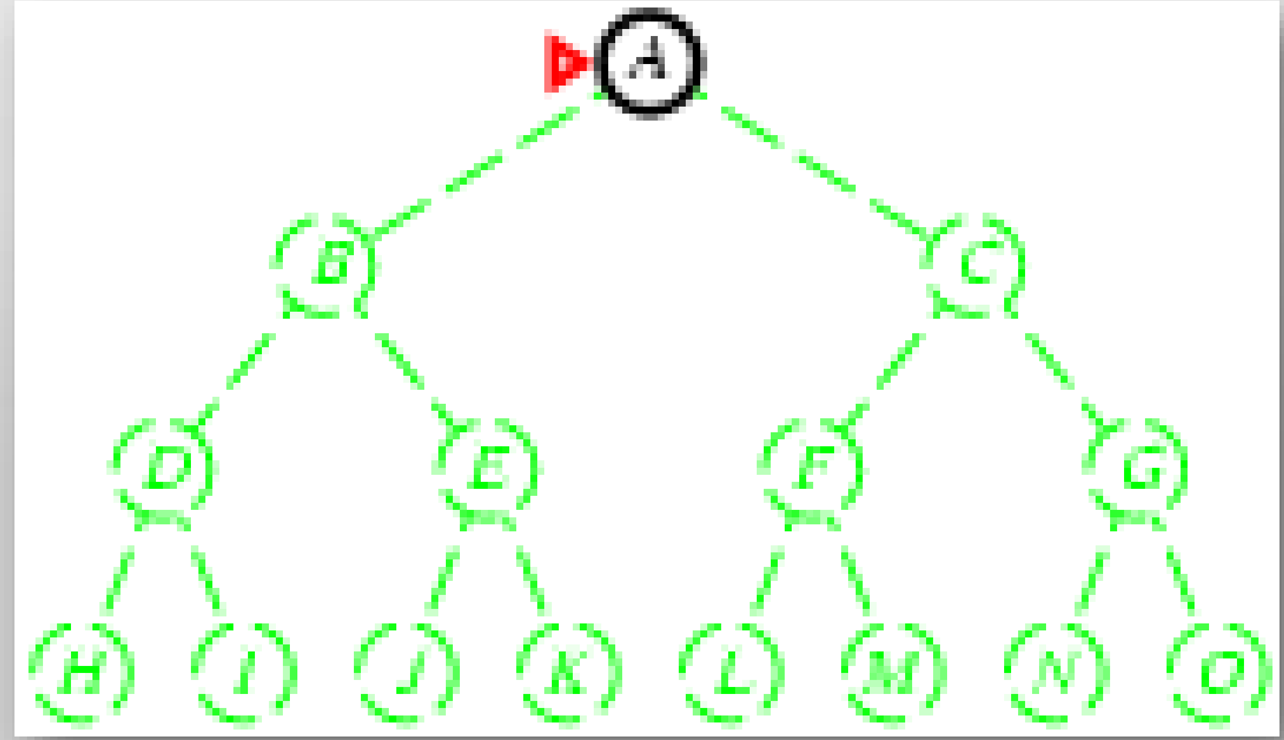
Depth-first search and the problem of memory

- Depth-first search always expands the deepest node in the frontier first
- It could be implemented where the evaluation function is the negative of the depth
- Search proceeds immediately to the deepest level of the search tree, where the nodes have no successors
- Search then “backs up” to the next deepest node that still has unexpanded successors

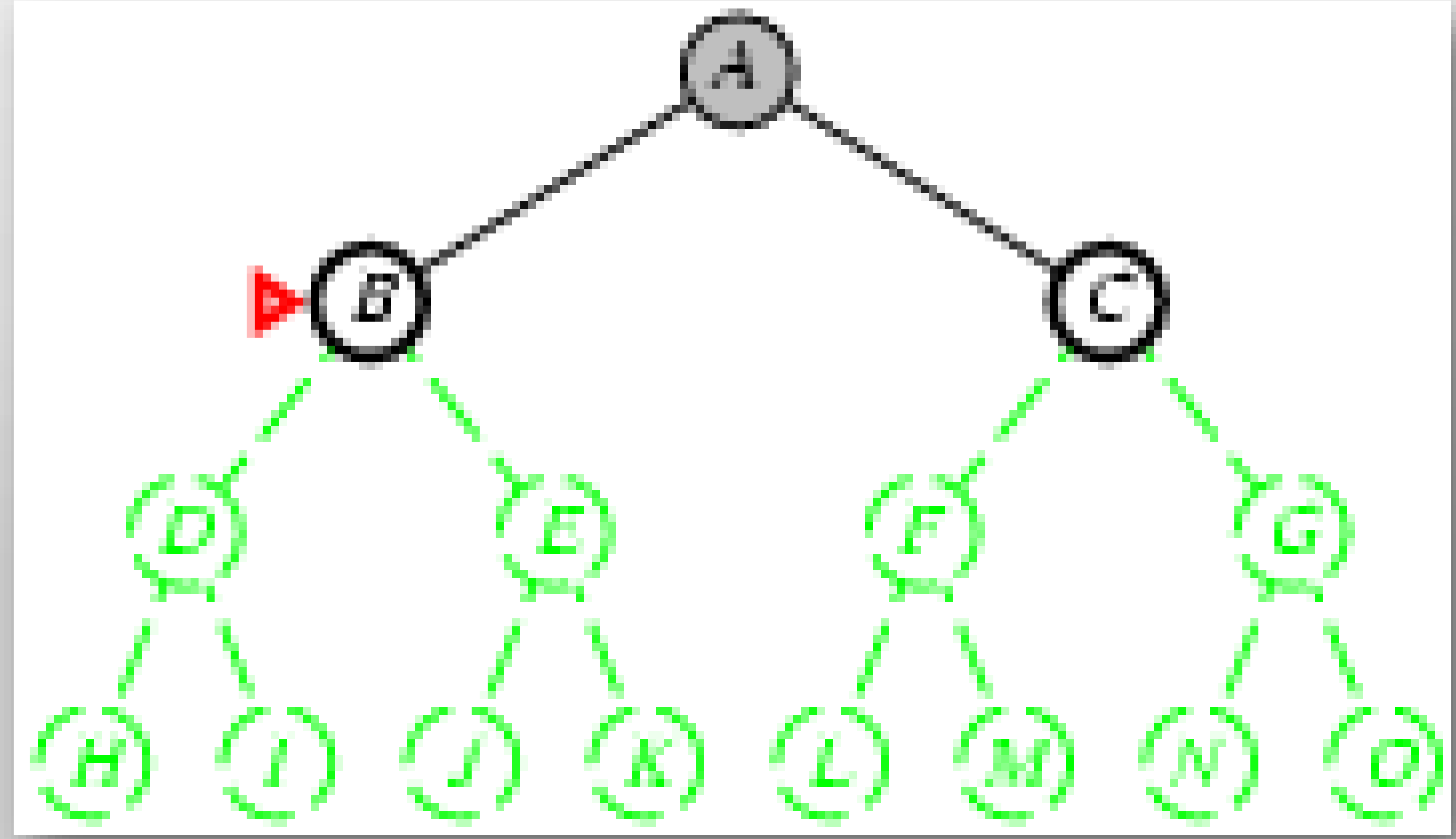
Depth-first search and the problem of memory

- Depth-first search is not cost-optimal; it returns the first solution it finds, even if it is not cheapest

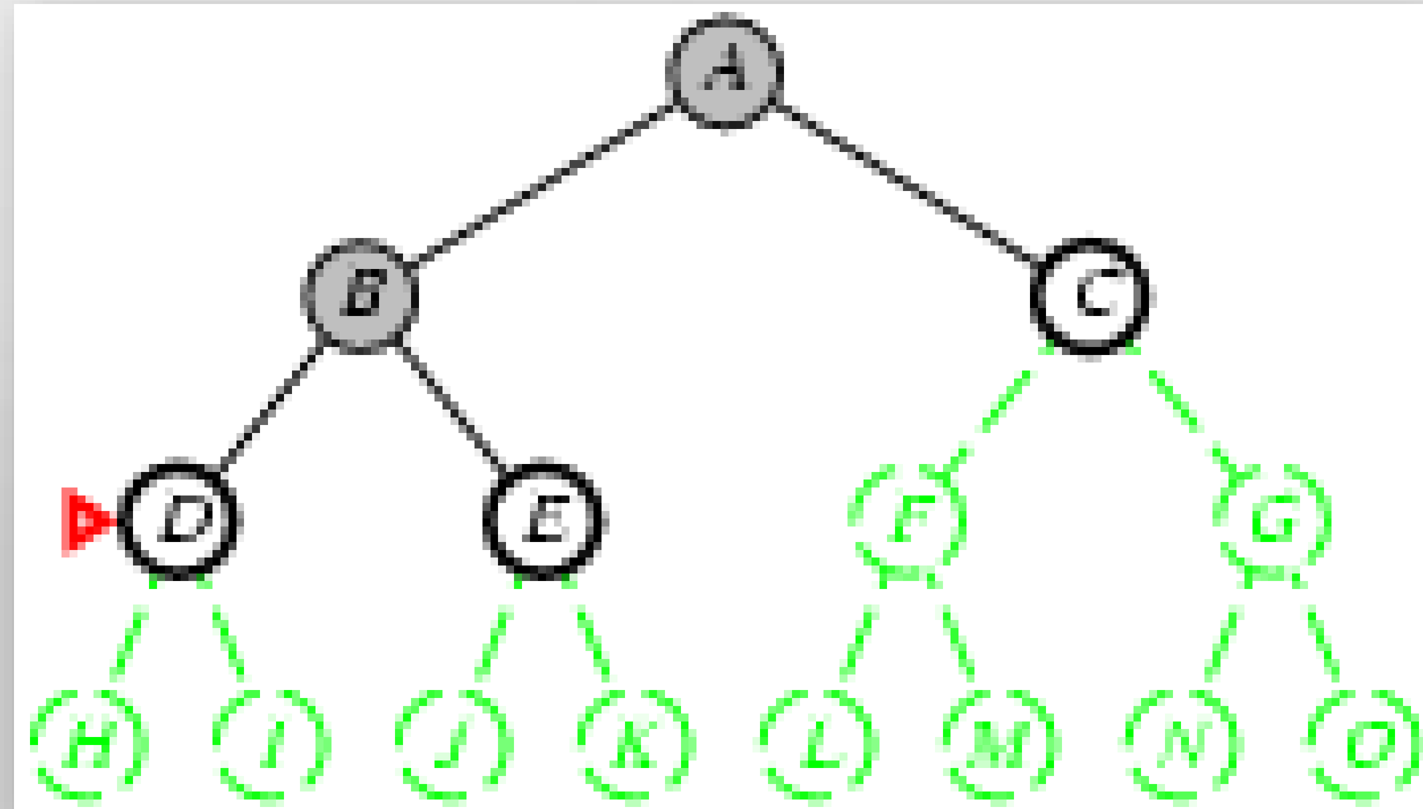
Depth-first search and the problem of memory



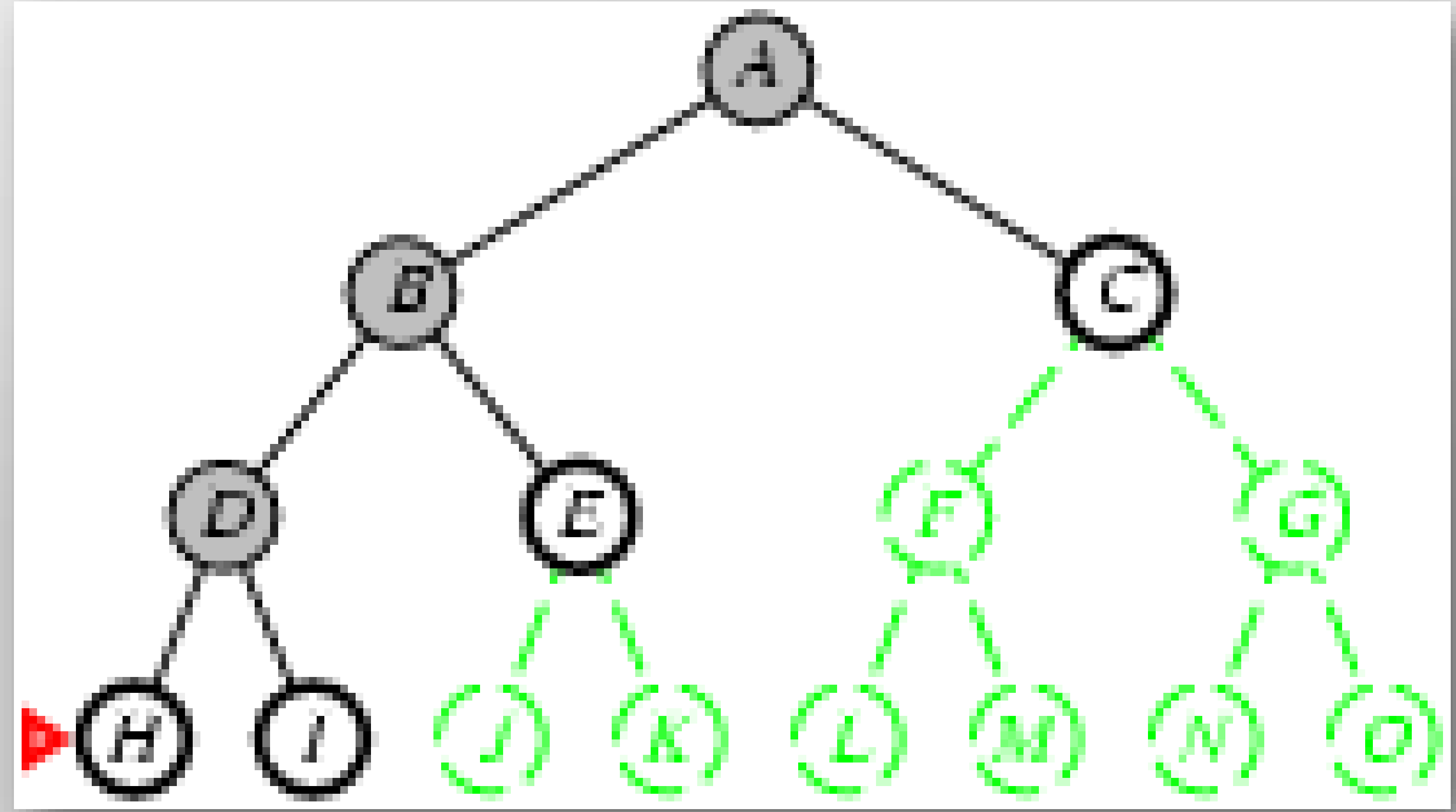
Depth-first search and the problem of memory



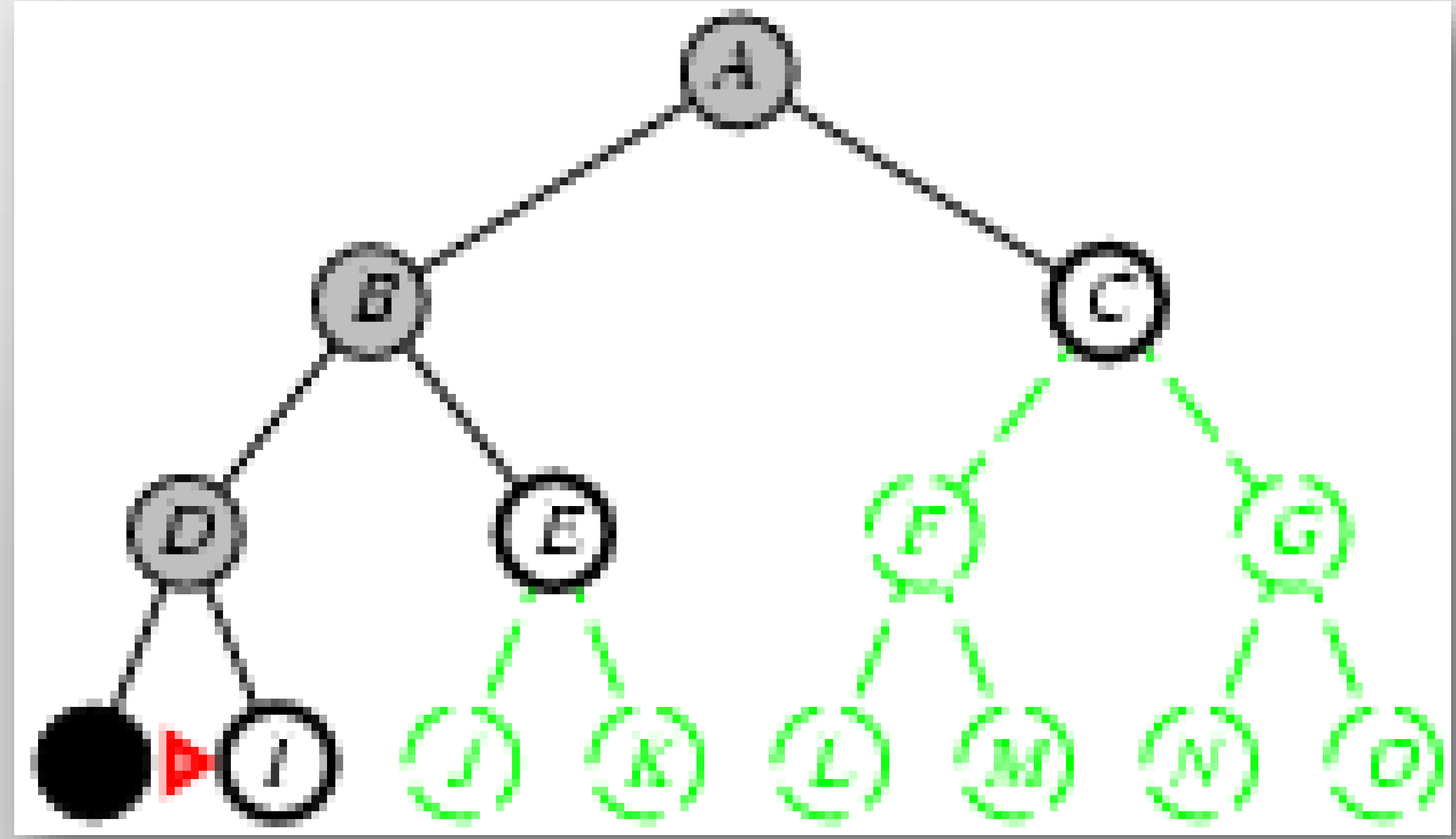
Depth-first search and the problem of memory



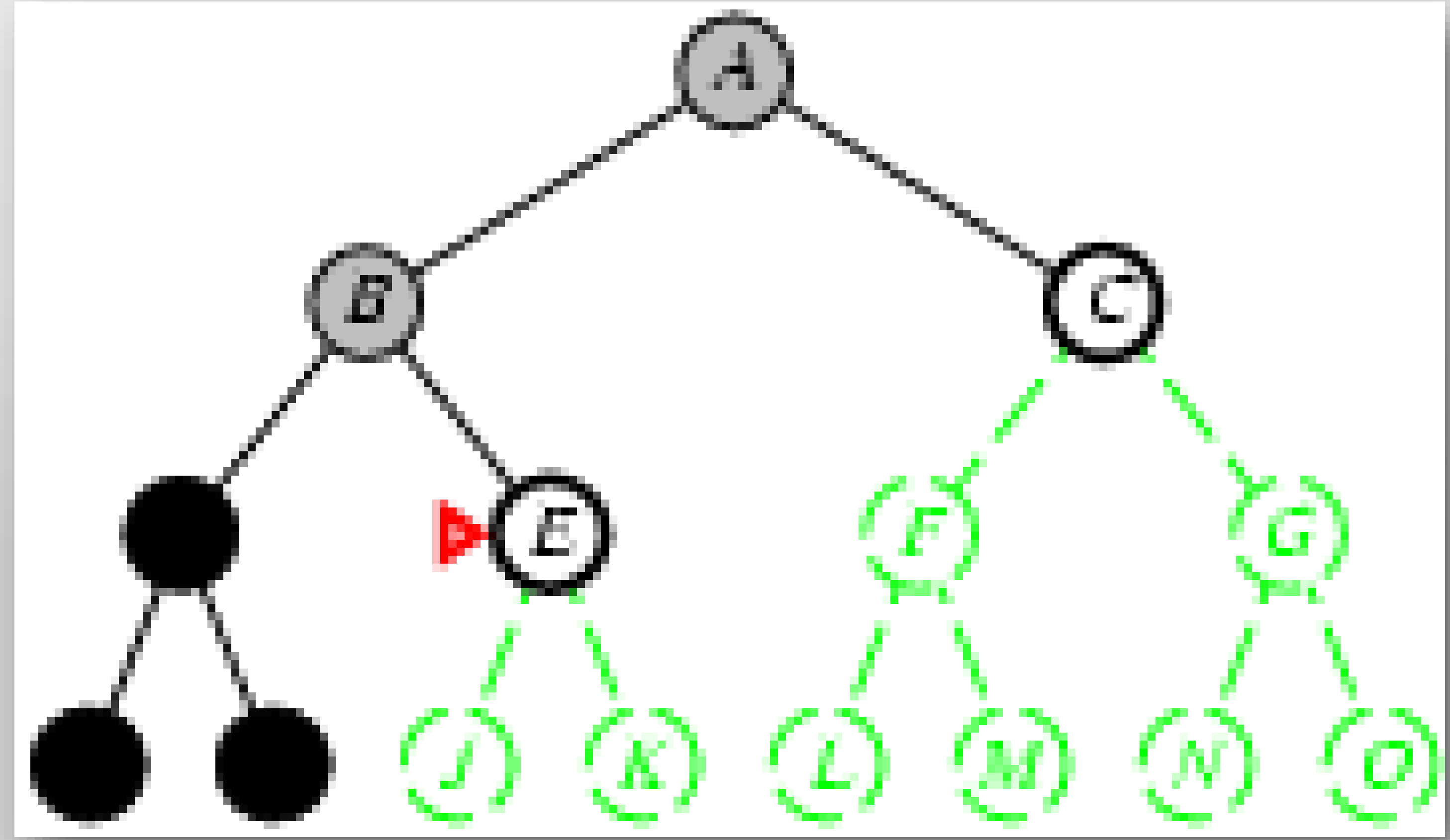
Depth-first search and the problem of memory



Depth-first search and the problem of memory

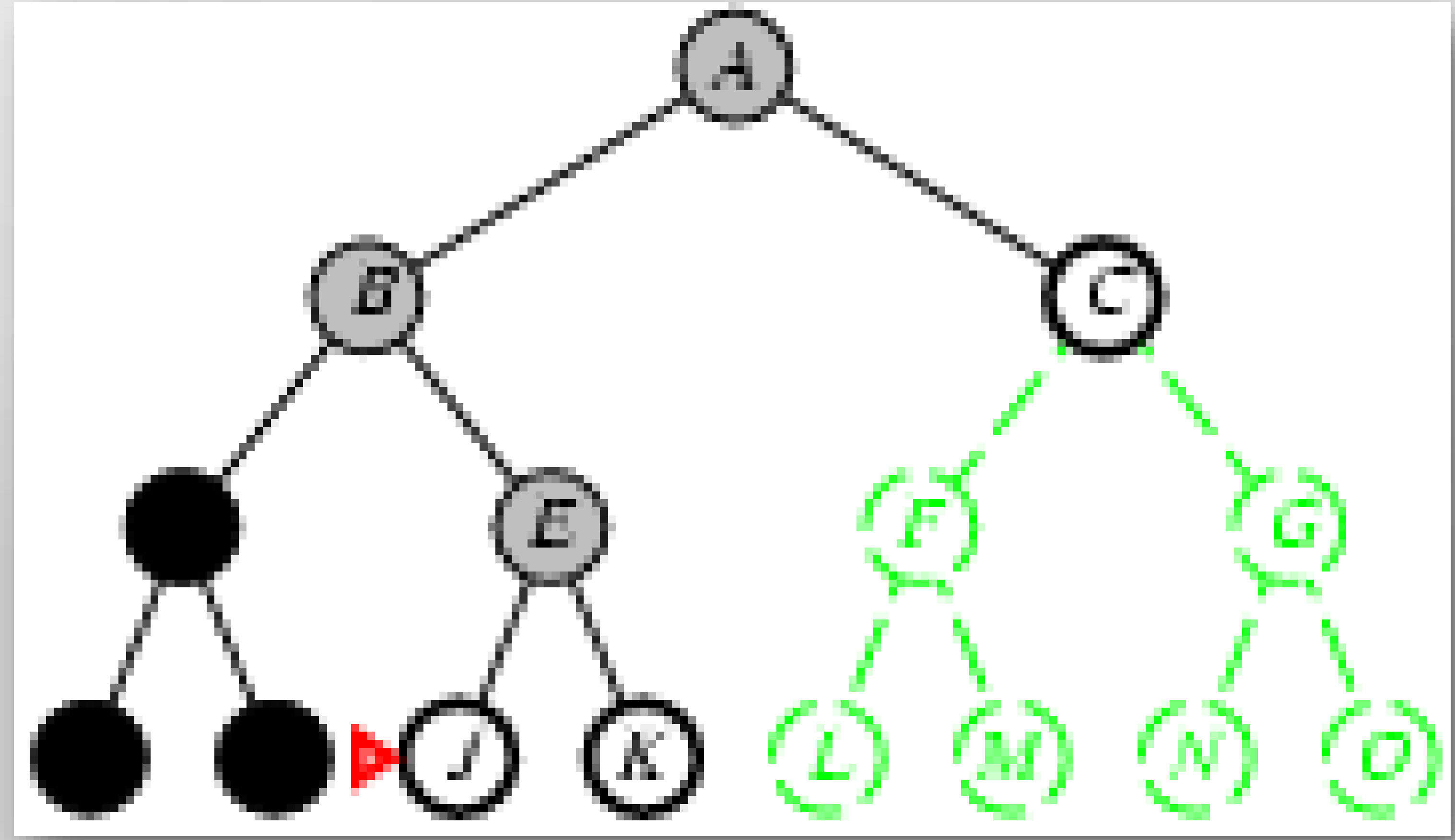


Depth-first search and the problem of memory

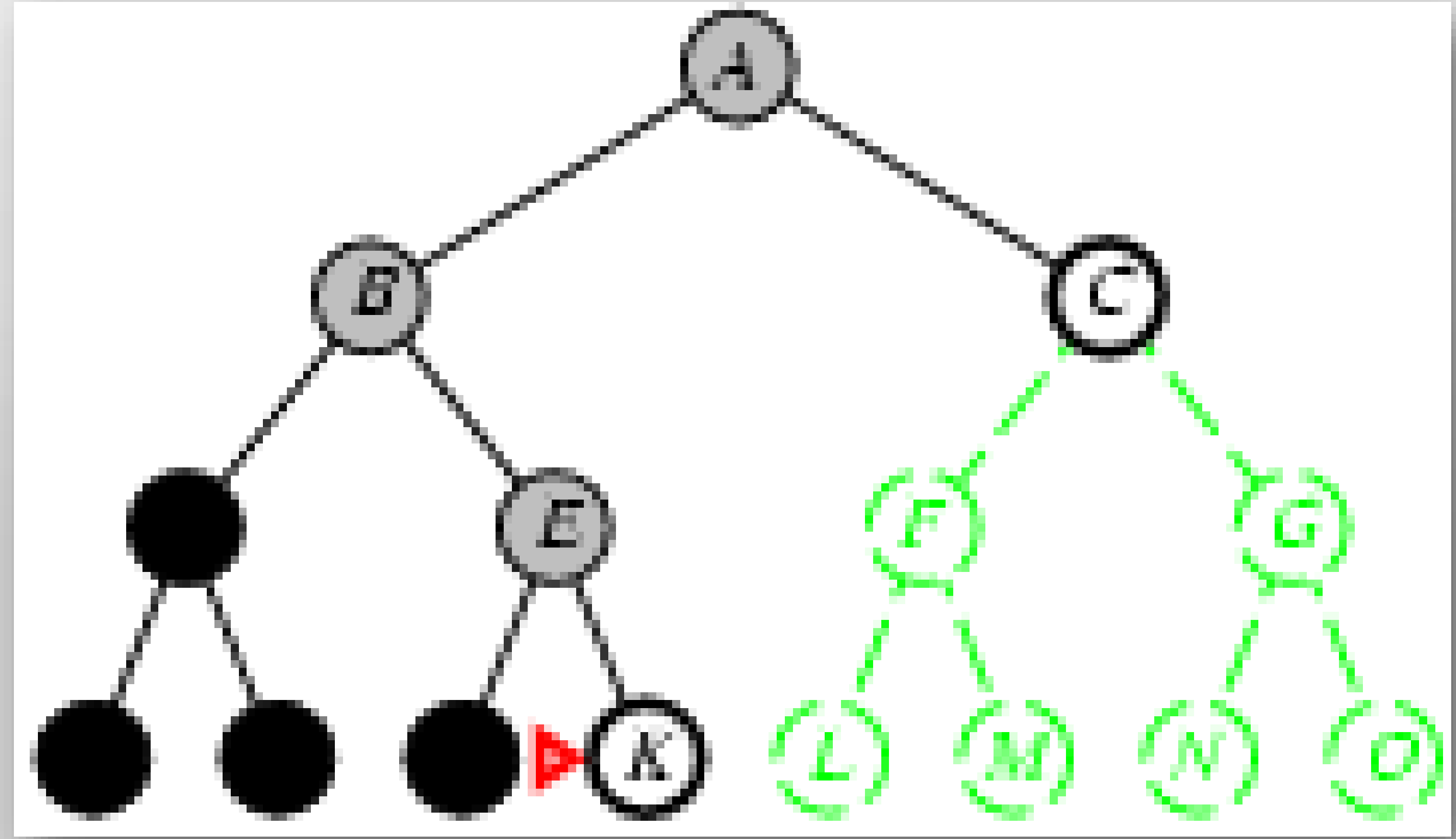


NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

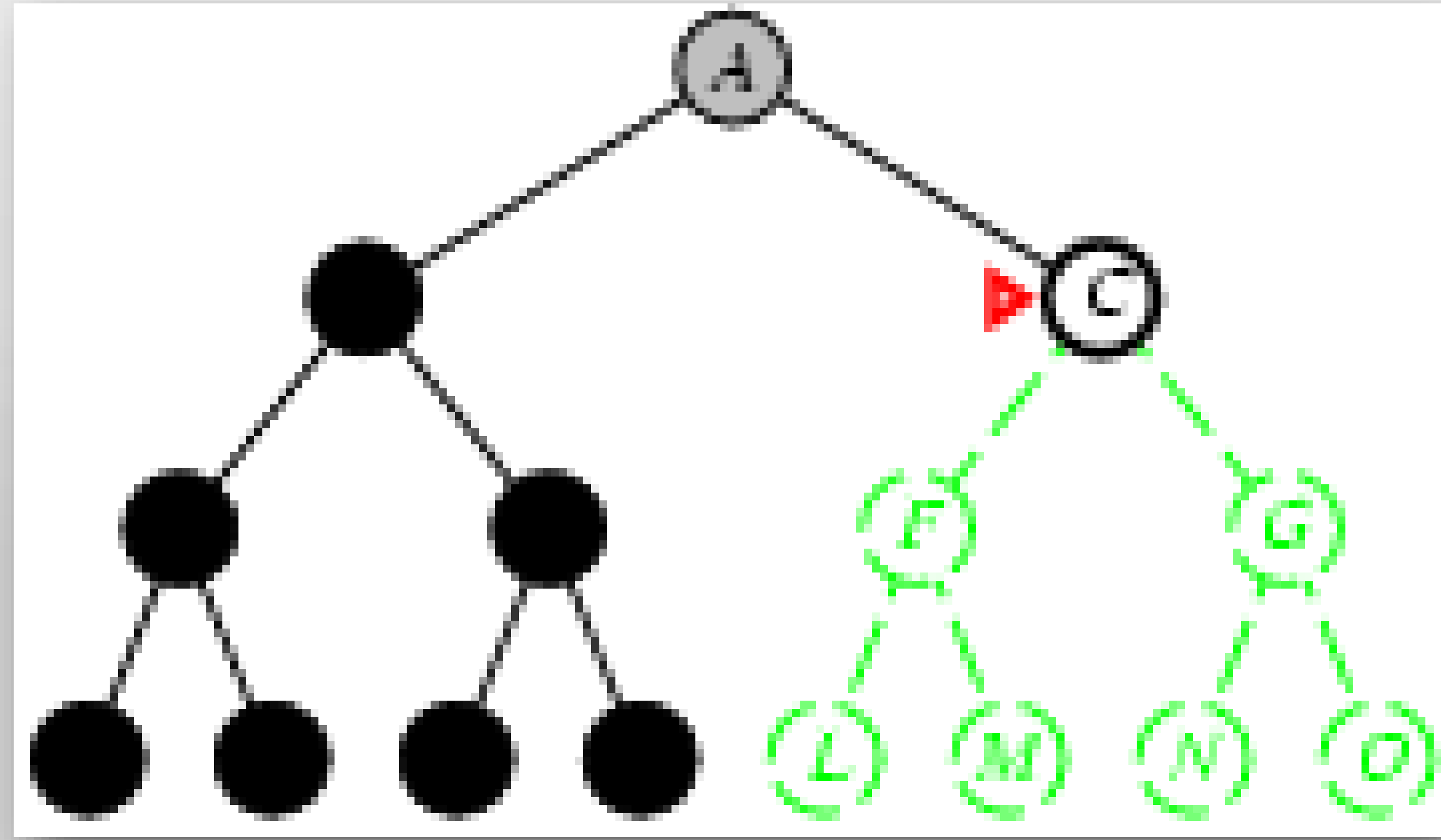
Depth-first search and the problem of memory



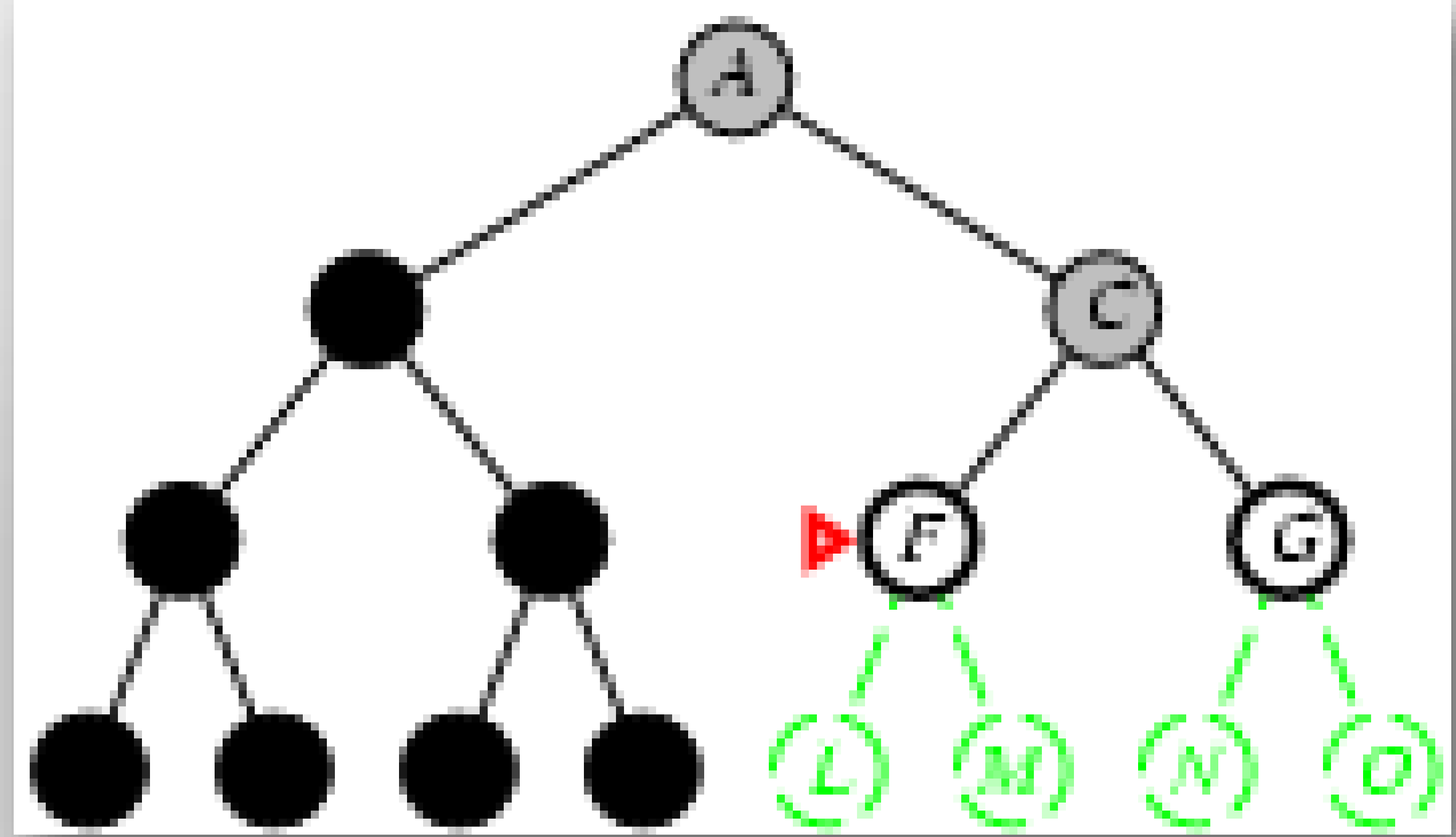
Depth-first search and the problem of memory



Depth-first search and the problem of memory

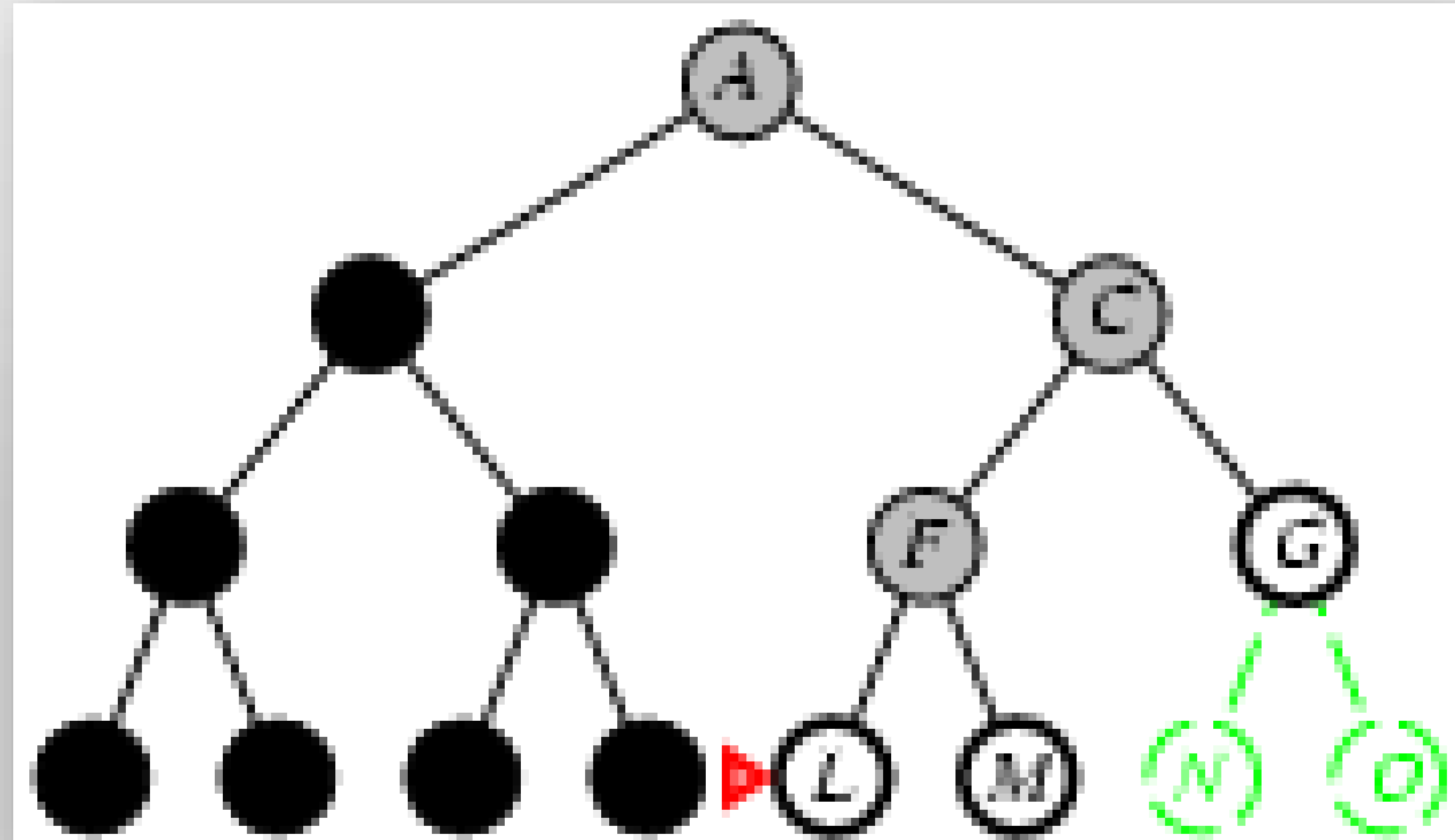


Depth-first search and the problem of memory



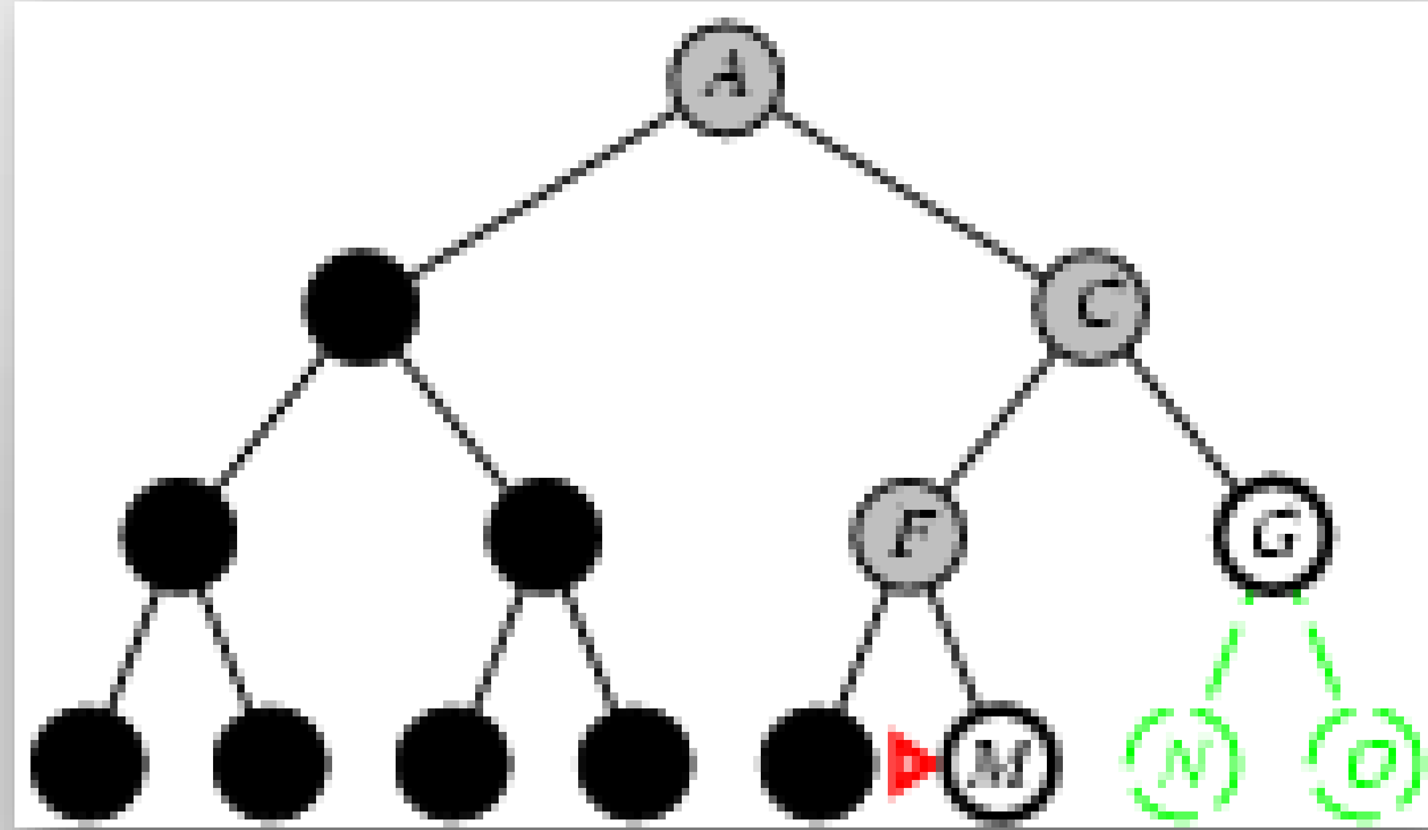
NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Depth-first search and the problem of memory



NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Depth-first search and the problem of memory



Depth-first search and the problem of memory

- For finite state spaces that are trees it is efficient and complete
- For acyclic state spaces it may end up expanding the same state many times via different paths, but will (eventually) systematically explore the entire space
- In cyclic state spaces it can get stuck in an infinite loop
- In infinite state spaces, depth-first search is not systematic: it can get stuck going down an infinite path, even if there are no cycles

Depth-first search and the problem of memory

- Depth-first search is incomplete
- For problems where a tree-like search is feasible, depth-first search has much smaller needs for memory
- For a finite tree-shaped state-space, a depth-first tree-like search takes time proportional to the number of states, and has memory complexity of only $O(bm)$ where b is the branching factor and m is the maximum depth of the tree

Backtracking search

- Use less memory than depth-first search
- Only one successor is generated
- Space complexity is $O(m)$
- Variation: successor is generated by modifying current state
- Memory requirement: one state, $O(m)$, actions

Depth-limited and iterative deepening search

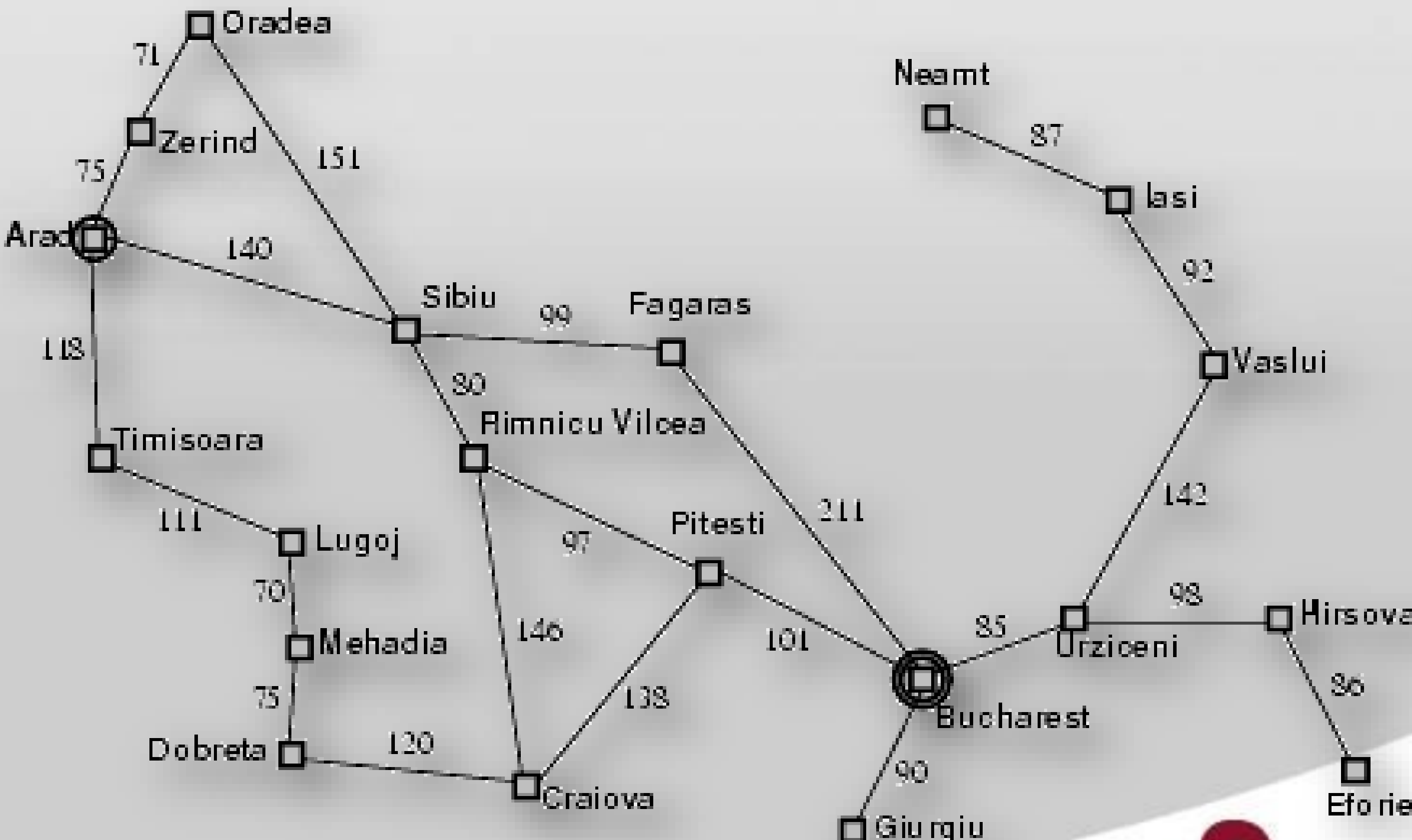
- To keep depth-first search from wandering down an infinite path we perform depth-first search to a predetermined depth-limit ℓ
- Nodes on depth of ℓ have no successors
- Search is incomplete if $\ell < d$ and non-optimal if $\ell > d$
- Time complexity: $O(b^{\ell})$
- Space complexity: $O(b\ell)$
- Depth-first search is a special case with

  $\ell = \infty$ (infinite)

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
INSTITUTIONAL OFFICE

Depth-limited and iterative deepening search

- Background knowledge of problem can help to determine ℓ



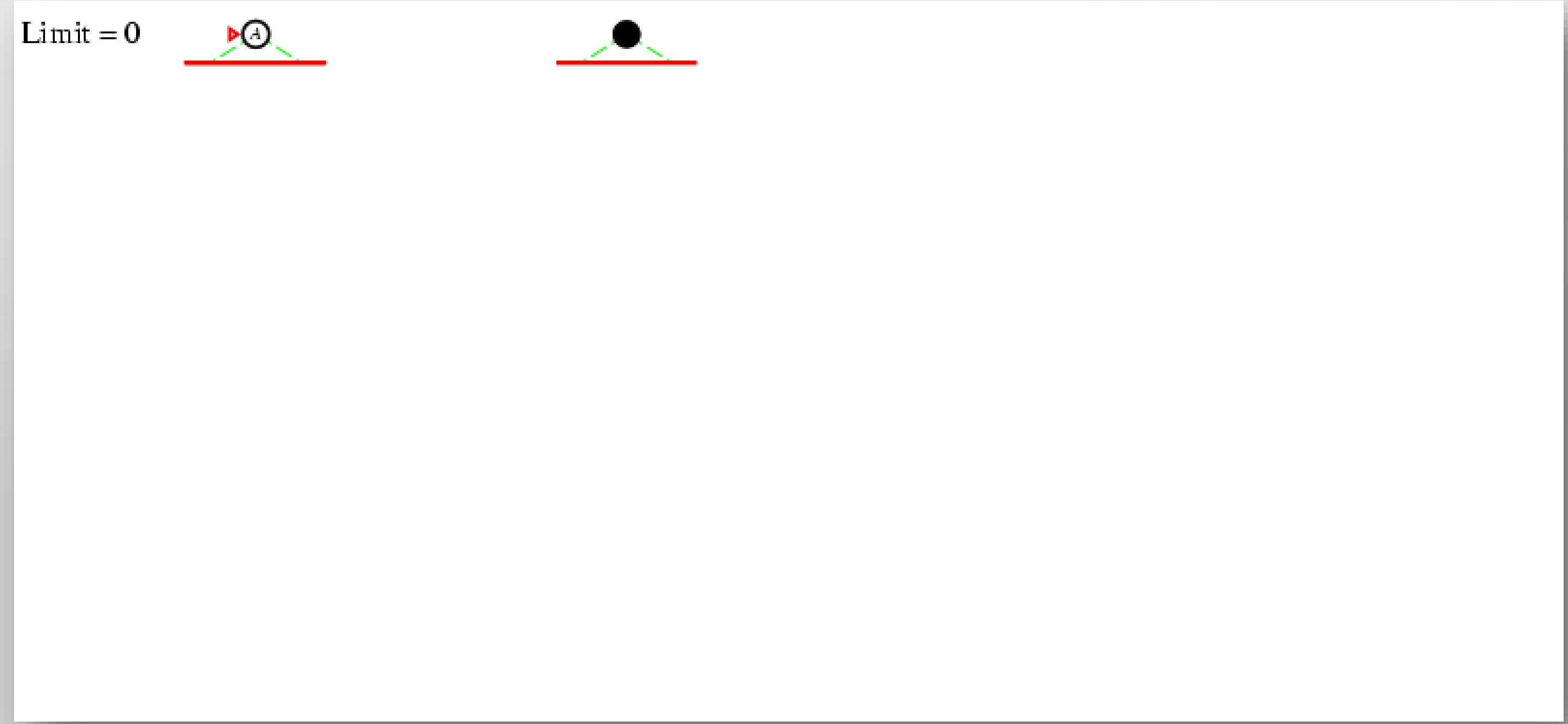
Depth-limited and iterative deepening search

- Iterative deepening search is used in combination with depth-first search
- Determines best depth limit ℓ
- Increase depth gradually, first 0, then 1, then 2, etc., until goal is found
- Combine properties of depth-first search and breadth-first search
- Space complexity is $O(bd)$
- Time complexity is $O(b^d)$
- Complete if branching factor is finite

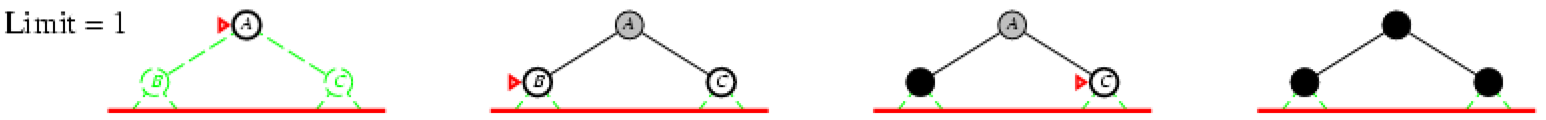
Depth-limited and iterative deepening search

- Optimal if path cost is non-decreasing function of depth of node
- Although states are generated more than once, strategy is very effective
- Strategy is similar to breadth-first search
- In general, iterative deepening is the preferred uninformed search method when the search state space is larger than can fit in memory and the depth of the solution is not known

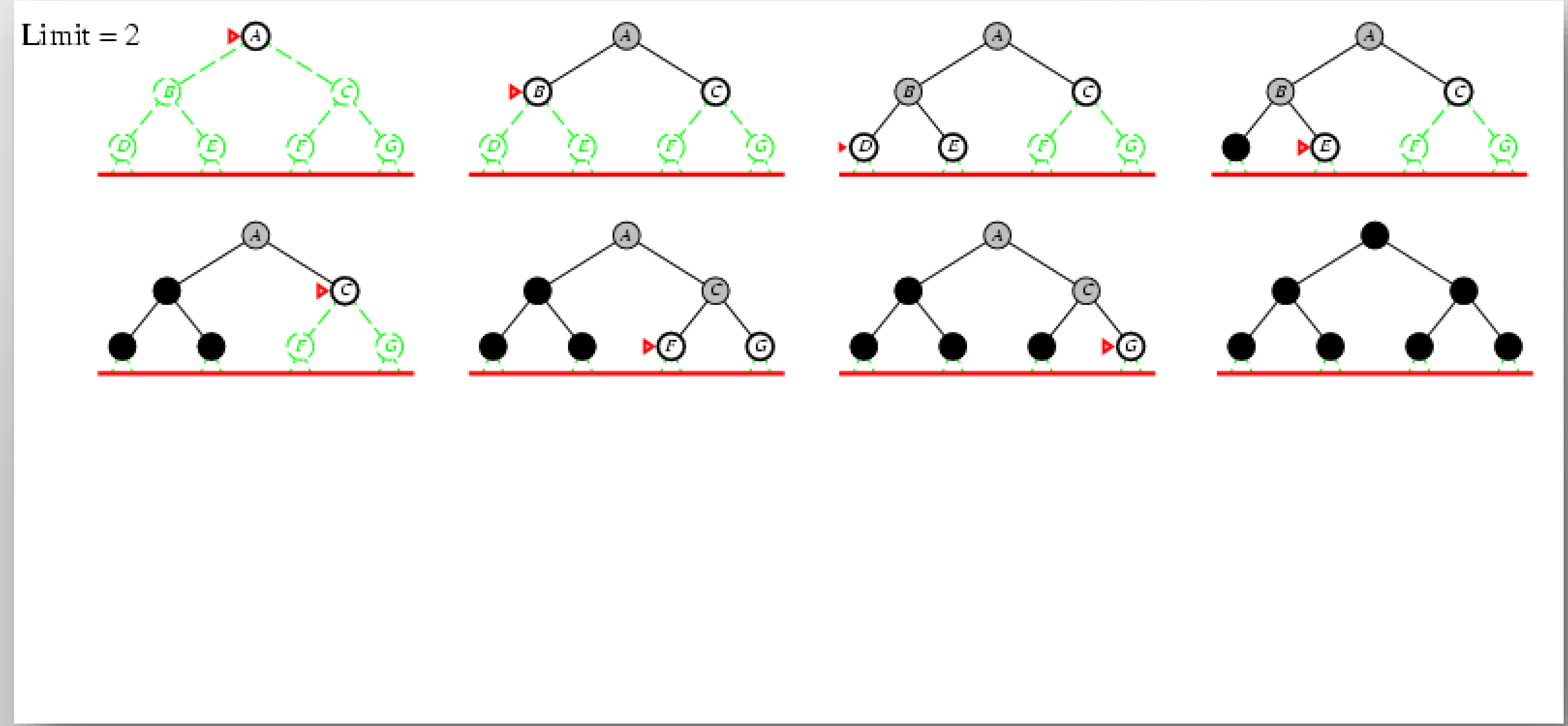
Depth-limited and iterative deepening search



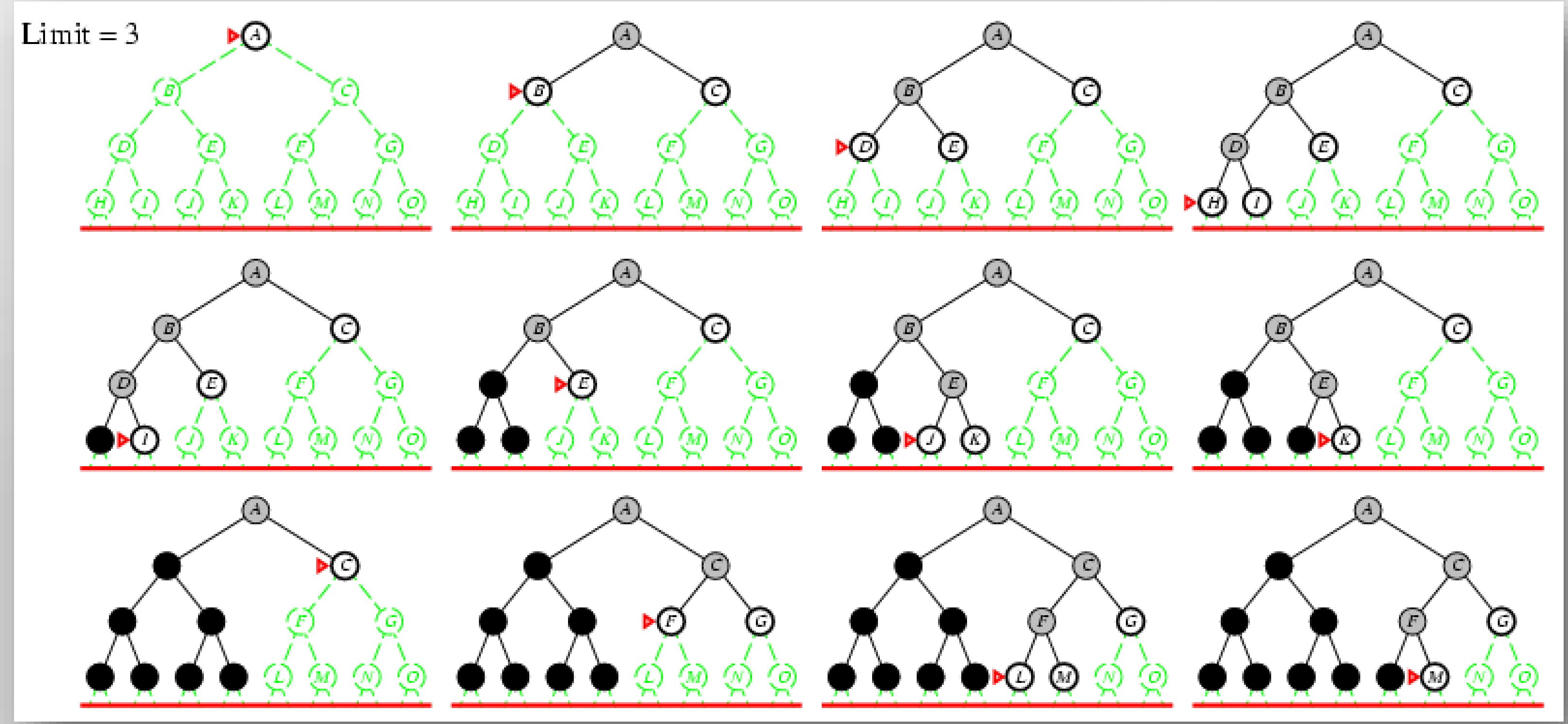
Depth-limited and iterative deepening search



Depth-limited and iterative deepening search



Depth-limited and iterative deepening search



Bidirectional search

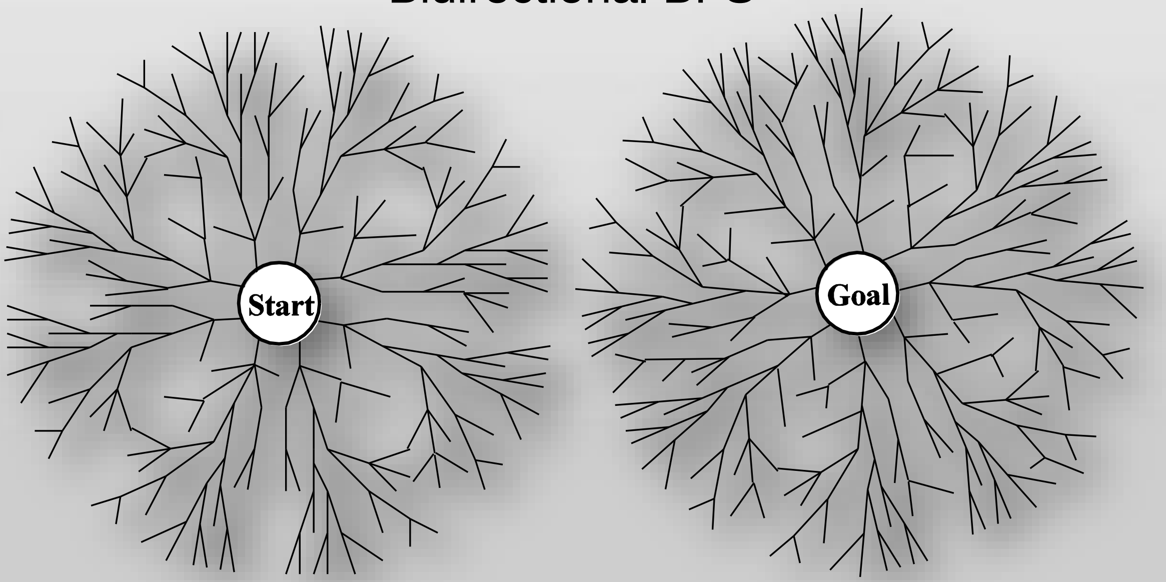
- Perform two simultaneous searches
  - One forward from initial state
  - Other backwards from goal
  - Stop when two searches come together in the middle
- We need to keep track of two frontiers and two tables of reached states, and we need to be able to reason backwards
- Time complexity is $O(b^{d/2})$
- Space complexity is also $O(b^{d/2})$

Bidirectional search

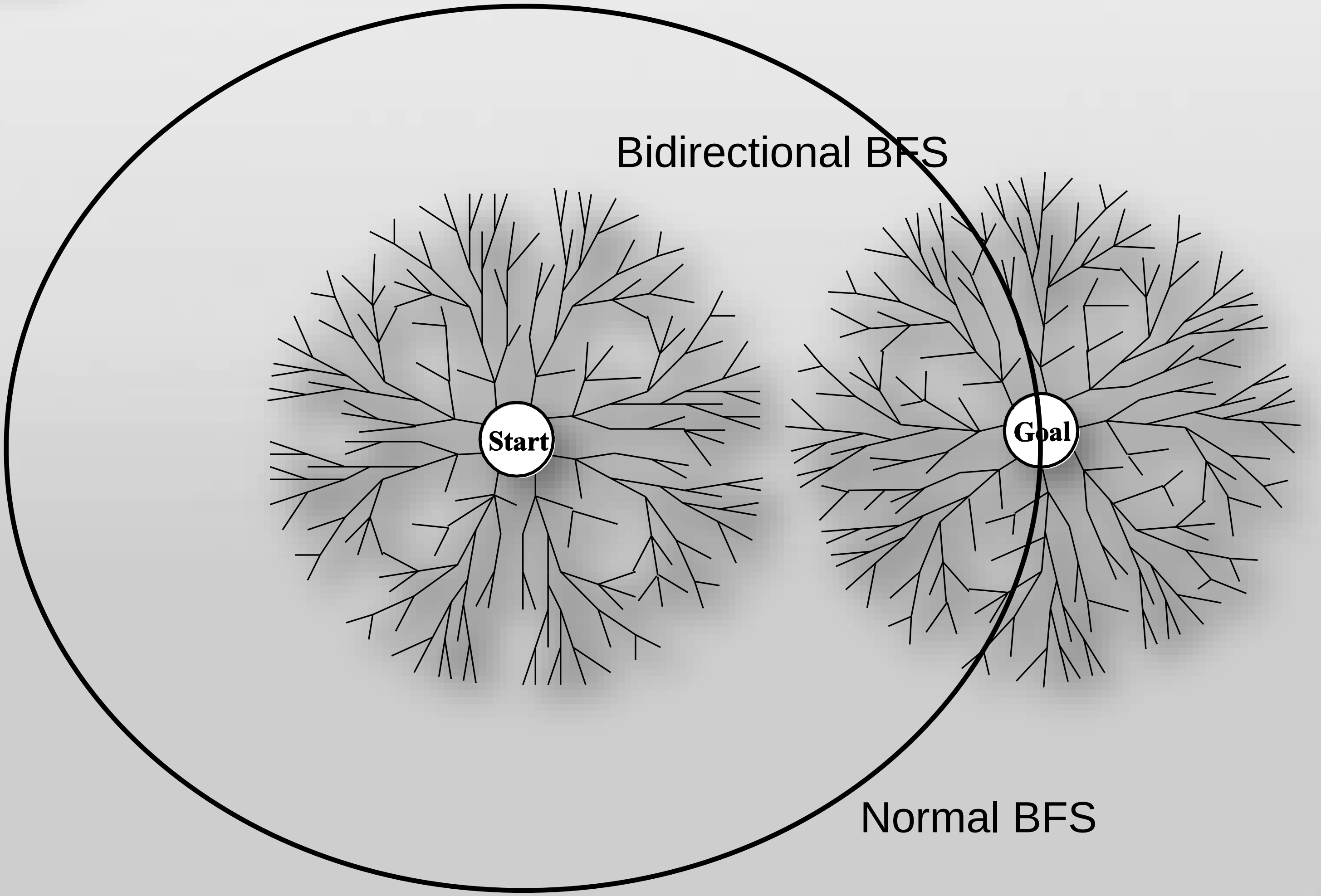
- Latter is main drawback
- Strategy is complete and optimal (for equal step costs) if both searches are breadth-first
- Search backwards sometimes difficult
  - More than one goal state
- Most difficult case when goal test is implicit description of goals

Bidirectional search

Bidirectional BFS



Bidirectional search



Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Assignment

- Study: Chapter 3.4 (Uninformed Search Strategies) of the AIMA e-book
- Self-study: Chapter 5 (Learning multiple weights at a time) of the Grokking Deep Learning e-book
- Theory Quiz 6: Chapter 3.4 (Uninformed Search Strategies) of the AIMA e-book
  - Thursday, 20 May 2021

Assignment

- Practical Quiz 5: Chapter 5 (Learning multiple weights at a time) of the Grokking Deep Learning e-book
  - Thursday, 20 May 2021
- Please study Appendix A.1 Complexity Analysis and O() Notation, at the end of the AIMA textbook